# MapReduce Assignments

注意：以下英文部分为英文课程<Introduction to data science>的 MapReduce 作业及要求，共有 6 个 problem，针对每个 problem 实现 python 代码。本课程的作业要求与该英文课程略有不同。

**不同点：**

1. 英文课程实现的 python 代码通过 python xxxxx.py input-data 命令来执行，是串行运算；
2. 本课程实现的 python 或 java 代码需通过 hadoop 及其 streaming 来执行，是并行运算；
3. MapReduce 本身是一种计算模型，是一种解决问题的思想方法，当用 python xxxxx.py input-data 来实现的时候是串行过程，当用 hadoop 来实现的时候是并行过程，只是集群化并行计算比串行计算能更有效地处理大数据；

**本课程作业要求：**

1. 同学们需完成全部 6 个 problem 的 MapReduce 程序，需能够在 hadoop 上运行，可以使用 python 或 java 语言编写；
2. 为了帮助同学们编写 Hadoop 版 MapReduce 代码，我们提供了符合英文课程作业要求的每个 problem 的 python 代码，同学们可以通过 python xxxxx.py input-data 命令来观察执行结果，同时可以通过参考这些代码来编写 Hadoop 版 MapReduce 程序；
3. 我们为同学们提供了一个 word count 的例子，分别使用 python 和 java 实现，同学们可以通过参考这个例子来实现其他 problem 的 Hadoop 版 MapReduce 程序；

Your python submission scripts are required to have a mapper function that accepts at least 1 argument and a reducer functionv that accepts at least 2 arguments. Your submission is also required to have a global variable named mr which points to a MapReduce object. If you solve the problems by simply replacing the mapper and reducer functions in wordcount.py, then this is guaranteed.

## Problem 1

Create an Inverted index. Given a set of documents, an inverted index is a dictionary where each word is associated with a list of the document identifiers in which that word appears.

# Mapper Input

The input is a 2 element list: [document_id, text]
document_id: document identifier formatted as a string
text: text of the document formatted as a string
The document text may have words in various cases or elements of punctuation. Do not modify the string, and treat each token as if it was a valid word. (That is, just use value.split())

# Reducer Output

The output should be a (word, document ID list) tuple where word is a String and document ID list is a list of Strings.

You can test your solution to this problem using books.json:

    python inverted_index.py books.json

You can verify your solution against inverted_index.json.

## Problem 2

Implement a relational join as a MapReduce query

Consider the query:

```
SELECT *
FROM Orders, LineItem
WHERE Order.order_id = LineItem.order_id
```

Your MapReduce query should produce the same information as this SQL query. You can consider the two input tables, Order and LineItem, as one big concatenated bag of records which gets fed into the map function record by record.

# Map Input

The input will be database records formatted as lists of Strings.

Every list element corresponds to a different field in it's corresponding record.

The first item(index 0) in each record is a string that identifies which table the record originates from. This field has two possible values:

    'line_item' indicates that the record is a line item.

    'order' indicates that the record is an order.

The second element(index 1) in each record is the order_id.

LineItem records have 17 elements including the identifier string.

Order records have 10 elements including the identifier string.

# Reduce Output

The output should be a joined record.

The result should be a single list of length 27 that contains the fields from the order record followed by the fields from the line item record. Each list element should be a string.

You can test your solution to this problem using records.json:

    python join.py records.json

You can verify your solution against join.json.

## Problem 3

Consider a simple social network dataset consisting of key-value pairs where each key is a person and each value is a friend of that person. Describe a MapReduce algorithm to count the number of friends each person has.

# Map Input

The input is a 2 element list: [personA, personB]
personA: Name of a person formatted as a string
personB: Name of one of personA's friends formatted as a string
This implies that personB is a friend of personA, but it does not imply that personA is a friend of personB.

# Reduce Output

The output should be a (person, friend count) tuple.
person is a string and friend count is an integer describing the number of friends 'person' has.
You can test your solution to this problem using friends.json:

```
python friend_count.py friends.json
```

You can verify your solution against friend_count.json.

## Problem 4

The relationship "friend" is often symmetric, meaning that if I am your friend, you are my friend. Implement a MapReduce algorithm to check whether this property holds. Generate a list of all non-symmetric friend relationships.

# Map Input

The input is a 2 element list: [personA, personB]
personA: Name of a person formatted as a string
personB: Name of one of personA's friends formatted as a string
This implies that personB is a friend of personA, but it does not imply that personA is a friend of personB.

# Reduce Output

The output should be the (person, friend) and (friend, person) tuples for each asymmetric friendship.
Note however that only one of the (person, friend) or (friend, person) output tuples will exist in the input. This indicates friendship asymmetry.
You can test your solution to this problem using friends.json:

```
python asymmetric_friendships.py friends.json
```

You can verify your solution against asymmetric_friendships.json.

## Problem 5

Consider a set of key-value pairs where each key is sequence id and each value is a string of nucleotides, e.g., GCTTCCGAAATGCTCGAA....
Write a MapReduce query to remove the last 10 characters from each string of nucleotides, then remove any duplicates generated.

# Map Input

The input is a 2 element list: [sequence id, nucleotides]
sequence id: Unique identifier formatted as a string
nucleotides: Sequence of nucleotides formatted as a string

# Reduce Output

The output from the reduce function should be the unique trimmed nucleotide strings.
You can test your solution to this problem using dna.json:

```
python unique_trims.py dna.json
```

You can verify your solution against unique_trims.json.

## Problem 6

Assume you have two matrices A and B in a sparse matrix format, where each record is of the form i, j, value. Design a MapReduce algorithm to compute matrix multiplication: A x B

# Map Input

The input to the map function will be matrix row records formatted as lists. Each list will have the format [matrix, i, j, value] where matrix is a string and i, j, and value are integers.
The first item, matrix, is a string that identifies which matrix the record originates from. This field has two possible values:

‘a’ indicates that the record is from matrix A
‘b’ indicates that the record is from matrix B

# Reduce Output

The output from the reduce function will also be matrix row records formatted as tuples. Each tuple will have the format (i, j, value) where each element is an integer.
You can test your solution to this problem using matrix.json:

```
python multiply.py matrix.json
```

You can verify your solution against multiply.json.